International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-2, Issue-4, July-2014

A Novel Solution to the Synchronization Problem of CAN

Shilpa Mayannavar, Ashwini Desai, Uday V. Wali

***Abstract*— One of the main problems in synchronous digital system design is synchronization of different clocks in the system. Obviously, this problem is always present in CAN bus related equipment. The problem is more acute in CAN compared to other high speed serial protocols like I2C and SPI. Speed of communication is pre-negotiated but synchronization is provided by the data frame. No separate clock line is provided.**

We have implemented a CAN transceiver that can automatically detect the speed and synchronization from the data frame. We have approached the problem in two distinct steps. First step is to identify the baud rate. In the following step, which is applied to every frame independently, the sampling time is adjusted with respected to the frame clock. The method also adjusts the sampling rate to reduce the number of samples required to sense the input signal. We also use a novel bit-central-sampling scheme to increase the reliability of data reading.

***Index Terms*—CAN (Controller Area Network), Frame synchronization, I2C (Inter Integrated Circuits), SPI (Serial Peripheral Interface)**

I. INTRODUCTION

I2C and SPI are the most commonly used serial communication protocols. They both are synchronous as they have their own clock. On the other hand, CAN is an asynchronous serial communication protocol used for communication between microcontroller and other peripherals. At any given time, one device acts as transmitter and other all devices will act as receivers. They all share a common bus, as shown in Figure1 [2].

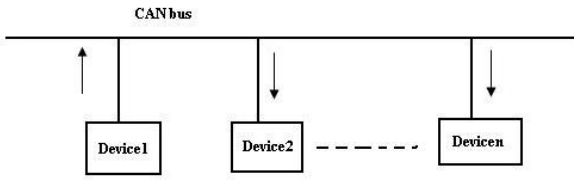

Figure 1: CAN bus sharing

Manuscript received July 23, 2014.

Ms. Shilpa Mayannavar, M.Tech. 4th semester student (VLSI Design and Embedded System), KLE DR MSS CET, Belgaum, India, 8431478910., (e-mail: mayannavar.shilpa@gmail.com).

Prof. Ashwini Desai, ECE Department PG section, KLE DR MSS CET, Belgaum, India, 9448230615., (e-mail: ashwini_pri@yahoo.co.in).

Dr. Uday Wali, Professor ECE Department PG section, KLE DR MSS CET, Belgaum, India, 9972638499., (e-mail: udaywali@rediffmail.com).

Here, Device1 is acting as a transmitter and other devices are receivers. As CAN is an asynchronous protocol the devices on the CAN bus should use an independent clock and mechanism for synchronization. Unlike in the SPI and I2C serial communication protocol, CAN does not have any separate clock, the data itself contains the clock. The data can arrive at any time with respect to internal clock; it can be on the rising edge of internal clock, falling edge or any where between the rising and falling edge. Essentially, we are unaware of the arrival timing of the data. Therefore, it is necessary to sample and interpret the incoming signal at a speed several times the baud rate. It is therefore reasonable to assume that the internal clock will be running at very high speed compared to the data that is coming in.

II. ORGANIZATION OF PAPER

Section-I gives an introduction of the Synchronization in communication protocols. Section-III defines the problem statement, section-IV describes the proposed approach for a solution, section-V describes implementation and section-VI shows some simulation results of the proposed method. Section-VII has conclusion and a brief description of scope for future work.

III. PROBLEM DEFINITION

As the CAN bus is an asynchronous message transfer protocol, it does not have a separate clock but the data frame contains the clock within itself. This clock has to be identified and synchronized with the internal clock of the controller. As the incoming signal has to be sampled at multiple times in a clock, controller's internal clock will be running at much high speed than the CAN input speed. Further, the CAN input may arrive at any time of the clock and hence there is no explicit synchronization between the internal clock and the CAN data coming in. This situation is presented in Figure 2 below [3].

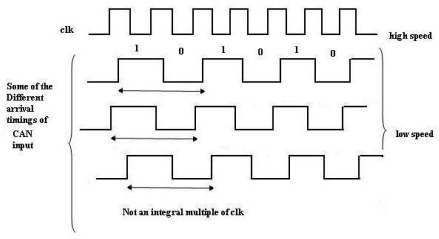

Figure.2: Problem statement of Synchronization in CAN

Note that the proposed method uses the preamble to achieve

synchronization. Therefore the synchronization must be achieved before the actual data enters.

Both frequency and phase synchronization is necessary to ensure reliable transfer of data. However, small differences in frequency and phase will always be present. As the CAN packets are small, small errors in synchronization are well tolerated. When the receiver is correctly synchronized, we can use fewer samples per clock, reducing the load on the processor of the controller. It will also increase reliability of received signal, assuming that there are no bus errors.

IV. PROPOSED METHOD

We make an assumption that CAN transmission occurs at any one of the predefined speeds. This is not unusual for most of the other serial communication protocols. Some of the commonly used baud rates are 4800, 9600 … 115200, etc. For a given internal clock speed and CAN baud rate, we can define table that contains a ratio of these two clocks.

Table 1. Look-up table for different clock speed

Clock speed	CAN Baud rate	Clock Cycles per CAN bit
20MHz	1Mbits	20.00
	115200	172.00
	9600	2083.33
10MHz	115200	86.00
	9600	1041.66
5MHz	115200	43.00
	9600	520.80
1MHz	9600	104.20

If we use some mechanism to count the number of clocks within one bit of CAN data input, we can look-up the speed ratio table and compute the baud rate. Note that the table will have known ratios stored as integer values. There could be small errors due to this integer conversion. However, they also represent a theoretical limit on achievable accuracy. Only way to improve beyond that is to change the controller clock to a higher value. So, the first part of the problem can be easily solved using a look-up table approach [2].

Once the problem of identifying the speed (auto-baud) is solved, we need to look at the phase of the signal. The task is to identify when the signal can be reliably sampled to get correct data. We use a sliding window concept to sense the correct phase. The width of sensing window is fixed by the controller clock/baud rate ratio table, discussed in the previous paragraph. We have to identify where a clock transition occurs, and initiate a window on that edge. Slow transitions of CAN signal can cause ambiguities in identifying the exact clock edge. However, this can be improved over multiple synchronization pulses transmitted during the preamble phase of the CAN data frame (Figure 3) [2]. This consists of a series of 0s and 1s, followed by Start of Frame (SOF) indicated by 1 following 1 in next clock. We have to estimate the CAN bit width during the preamble phase and use it to sample the SOF and subsequent data bits.

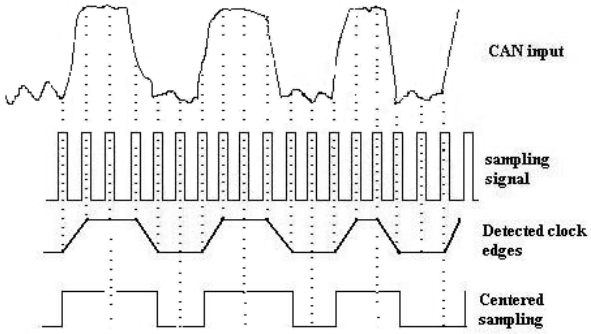

Figure 3: Preamble phase of CAN data frame

We have used an *up down counter with preload* to know the number of clocks within one bit of CAN data input. The counter is implemented using cascaded JK flip-flops. Output of each flip-flop can be read out in parallel to get the count. The counter is started on an edge of the CAN input. The counter is read on the next edge. This value is stored in one of a set of temporary registers. The process is repeated for next bits. We use a minimum of 3 bits of preamble to estimate the bit width. The bit widths (in terms of sampling clock) may vary by one or two counts. Maximum of these values is generally correct bit width. Assuming that the transmitter is using one of the standard bit rates, the number of clocks per CAN bit width can be precomputed. These values, indicated in Table 1 are stored in a lookup table. By comparing the estimated bit width with values in the lookup table, we can select the correct baud rate. The trick to achieve auto-baud, therefore, is to measure the difference from the table value and not to try to match the value exactly. The one with least difference will decide the baud rate. Notice that this '*look up and difference'* will effectively ignore the errors in signal received during synchronization phase but reliably estimate the baud rate.

Once the baud rate is selected, synchronization problem is reduced to sliding window problem to match the edge of clock. However, this information is also available in the data we have already sampled and hence no extra sampling is required [3]. Start of a bit on CAN bus needs to be established correctly to be able to read the input. Input jitter with respect to clock is estimated in terms of clocks to estimate the correct edge. Down counter is started from this clock with number of clock counts corresponding to estimated baudrate. The counter will self-preload after the counter value reaches zero. This will synchronize the clock with the incoming CAN input correctly.

Further, Samples are taken on multiple, consecutive clock periods, rather than uniformly spread across the CAN bit width. This is because the data is likely to be correct at the center of the bit width rather than at the edges.

V. IMPLEMENTATION

The proposed method is implemented in ICARUS Verilog using Verilog language and the simulation results are viewed on a GTKWave platform. The block diagram for the

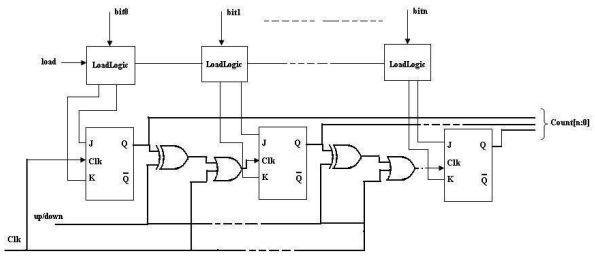

Figure 4: Block diagram of up/down counter with preload

up down counter is shown in Figure 4.

As it can be seen from the Figure 3, each bit of CAN input is loaded using the load logic into the JK flipflop. When load is set, parallel input (preload) is latched into the flipflops. Otherwise, serial data is shifted through the flipflops. The output of flip-flop is either incremented or decremented depending on the up/down signal. On the rising edge of the clock when the CAN input becomes high counter starts incrementing and the number of clocks will be counted till the CAN input goes low. This count is used to estimate the baud rate from a look-up table. Clock count based on the baud rate is used to set the count-down timer for next bits of CAN input [1]. In order to achieve a better synchronization, CAN bit width in terms of internal clock must be of the order of 10^3.

State diagram for CAN synchronization is as shown in Figure 5 below.

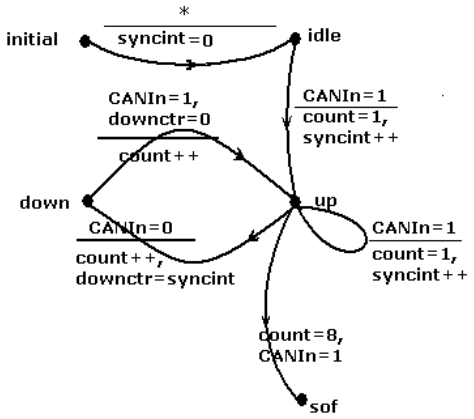

Figure 5: state diagram for CAN synchronization

There are mainly five states *initial*, *idle*, *up*, *down* and *sof*. All the variables are initialized in the *initial* state, assumed at the boot-up time. The state changes to *idle*, during a bootup procedure. CAN transceiver will be in this state most of the time, waiting for an event to occur on the CAN bus. The variable *syncint* indicates the number of clock pulses within one bit of CAN input (*CANIn*). When the internal clock is high and CAN input also goes high, then the state changes from *idle* to *up* and the counter starts incrementing. In the *up* state if the CAN input becomes low then the state changes to *down* and the counter increments and value of *syncint* is loaded into the down counter and the counter starts decrementing until it becomes zero. The state then changes to *up*. The process repeats till end of frame. When count becomes 8 on the positive CANIn, the state changes to sof (start of frame). Actual data (payload) is sampled from then onwards till a eof (end of frame) is received. Overall state transition diagram for receiving a frame is given in fig. 6.

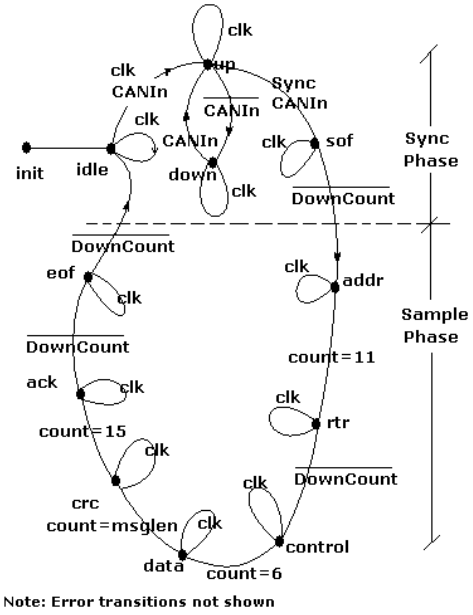

Figure 6: CAN Receiver

We have marked two distinct phases in the state transition diagram. The first phase, marked as sync-phase is already explained (fig. 5). The other phase, called the sample-phase reads the data when the down counter triggers the events. The transitions correspond to the CAN frame structure. Key issues here are reading the CAN control field which dictates the length of the data to be read. Number of cycles within *data* state is decided by this value. Logic to handle stuff bits on the transmitter side has been published by our group elsewhere [4]. A similar unit can be used on the receiver side as well. Hence, we have not described the details here.

Key points in realization of a CAN receiver are

i) Synchronization between host and device
ii) Interpretation of arbitration field
iii) Calculation of number of bytes from the control field
iv) Responding when RTR is set
v) Stuffbit logic
vi) Evaluation of CRC and matching with CRC within the frame
vii) Presentation of Ack bit by the receiver and
viii) Error handling

We have implemented all stages except error handling.

VI. SIMULATION RESULTS

The simulation results as viewed in GTKWave GUI are shown in Figure 7 and Figure 8.

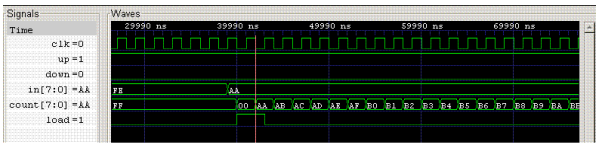

Figure 7: Simulation result of up/down counter

If the *load* signal is set, input is latched into the JK flip-flops. Counting starts when the load signal is pulled down. Counter value is incremented or decremented depending on *up/down* signal. On reset, maximum number is latched into the JK flip-flop followed by up/down counting. For example FFh for 4 bit counter (Figure 7).

The match 1 indicates that both bits (1 and 0) of the CAN input are approximately same (Figure 8). Then we can use this count value to set the timer for remaining bits [3].

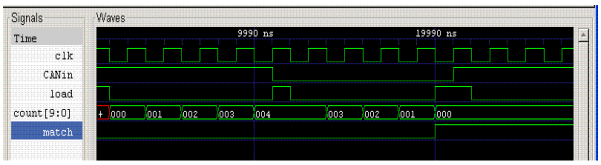

Figure 8: Simulation result of CAN synchronization

VII. CONCLUSION

We have implemented a bus synchronization module with automatic speed detection capability, for a CAN controller, in Verilog. Most of the design has been designed using structural representation, minimizing behavioral code that may limit the synthesizability. The device is capable of both transmitting and receiving CAN messages, confirming to the CAN standards. As it is a low level device, it can handle various CAN related protocols like DeviceNet from Rockwell/Allen Bradlez, CANOpen from CiA, etc transparently. As the device is available as a soft core, it can be integrated with soft core controller designs.

VIII. FUTURE WORK

We have worked on most of the modules required for CAN controller. We have also added auto-baud capability. Integration with SPI and I2C has also been published elsewhere [4]. However, many extensions to CAN device capabilities are required for use in the new building automation and other similar large networks. These works are in progress and will be published in due course of time.

ACKNOWLEDGMENT

The authors would like to thank C-Quad computers and KLE DR MSS CET Belgaum for providing all the facilities and support.

REFERENCES

[1] Digital electronics (Principles and Integrated Circuits) by Anil K. Maini WILEY PRECISE edition.

[2] U. V. Wali "Plug and Play CAN", Tutorial presented at International Embedded system Conference ESC JULY 2009, Bangalore.

[3] Shilpa Mayannavar, "A Novel solution to the Synchronization problem of CAN" M.Tech. Thesis submitted to VTU, June 2014.

[4] Design of CAN transmitter with an I2C interface, Anupama B and Uday Wali, International Journal of Computer Applications, Vol. 46, no. 21, ISBN 973-93-80868-65-5

[5] High speed CAN Transceiver, Microchip, available from web link http://ww1.microchip.com/downloads/en/DeviceDoc/21667E.pdf

[6] LPC2119/2129/2194/2292/2294 User Manual, NXP Semiconductor, http://www.nxp.com/acrobat_download/usermanuals/UM_LPC21XX_LPC22XX_2.pdf

[7] CAN specification 2.0, Rober Bosch, 1991, http://www.semiconductors.bosch.de/pdf/can2spec.pdf

Ms. Shilpa Mayannavar completed her Bachelor of Engineering with major in Electronics and Communication Engineering at SGBIT Belgaum and she has received VTU Gold medal for academic excellence in the field of ECE. Presently she is Pursuing Master of Technology in VLSI Design and Embedded systems at KLE DR MSS CET Belgaum.

Prof. Ashwini Desai completed her BE (E & CE) from GIT, Belgaum affiliated to Karnataka University, Dharwad and was awarded M.Tech. by VTU, Belgaum in the year 2005. She is currently pursuing her PhD, while working as Asst. Professor at KLE DR MSS CET Belgaum. She has presented her work in National and International conferences.

Dr. U. V. Wali is a Professor at KLE DR MSS CET Belgaum and Director at C-Quad computers, Belgaum. He was awarded Ph D by IIT Kharagpur in the year 1986 for his work on simulation of switched capacitor networks. He has published papers in many International conferences and journals. His research interests include VLSI physical design, testing and verification, software design and architecture, among others. He is a fellow of Institution of Engineers, India.